

Interprétation Compilation Projet

GOEHRY Martial
16711476

9 janvier 2022

Table des matières

1	Projet : Interpréteur BrainFuck	2
1.1	Analyse lexicale	3
1.1.1	Fonction : yylex	3
1.1.2	Fonction : yyparse	3
1.1.3	Gestion des block	6
1.2	Main	8
1.3	Exemple de sortie	8
2	Logiciel libre	10
2.1	Packaging	10
2.2	Man page	11
2.3	Installation désinstallation	15
3	Évaluation du cours	16
A	BrainFuck Interpréteur code complet	18

L'année dernière, je vous avais contacté pour vous proposer mon projet final, un interpréteur du langage BrainFuck. Je souhaitais également utiliser ce projet afin de valider le cours Logiciel Libre.

1 Projet : Interpréteur BrainFuck

Le langage Brainfuck est un langage assez simple ne contenant que 8 opérandes :

- > Déplacer le pointeur vers la droite
- < Déplacer le pointeur vers la gauche
- + Incrémenter la valeur de la case pointée
- Décrémenter la valeur de la case pointée
- . Afficher la valeur de la case pointée au format ASCII
- , Stocker la valeur donnée en STDIN dans la case pointée
- [Sauter au] correspondant si la valeur dans la case pointée est égale à 0
-] Sauter au [correspondant si la valeur dans la case pointée est égale à 0

L'idée de ce langage est de déplacer un pointeur le long d'un tableau d'au moins 30 000 cases. Toutes les cases de ce tableau sont initialisées à 0 au lancement de l'interpréteur. La valeur à l'intérieur de ces cases en comprise dans les limites des valeurs ASCII (entre 0 et 127).

Les opérandes [et] permettent de créer des block d'instruction. Je n'ai rien trouvé indiquant la possibilité de faire des imbrications de block. J'ai arbitrairement choisis d'autoriser l'imbrication jusqu'à 8 block.

L'une des limitations de mon interpréteur est que l'interprétation commence dès le retour chariot. De ce fait, une erreur est levée et l'interpréteur se ferme dans le cas où les block ne sont pas correctement renseignés.

1.1 Analyse lexicale

L'analyse lexicale est plutôt simple, je n'ai pas eu de règles de grammaire à renseigner.

1.1.1 Fonction : *yylex*

La fonction *yylex* est également assez simple. Seul les opérateurs autorisés sont acceptés, tous les autres char sont purement et simplement ignorés. Le retour chariot marque la fin de la ligne.

yylex retourne la valeur de *yylval* qui va être une des opérations autorisés.

Listing 1 – Variables globales

```
1 char *ops = "><+-.,]";           // Operateurs acceptes
```

Listing 2 – *yylex*

```
1 int yylex(void) {
2
3     // Recommencer a la fin de la ligne
4     if (ligne[indexligne] == '\n'){
5         return 0;
6     }
7
8     // Retirer tous les char qui ne sont pas des operateurs
9     while (!( strchr(ops, ligne[indexligne])) )
10         indexligne++;
11
12     // Recuperation de l'operateur
13     yynval = ligne[indexligne++];
14     return yynval;
15
16 }
```

1.1.2 Fonction : *yyparse*

La fonction *yyparse* est un peu plus conséquente. Un switch va permettre de traité chaque valeur possible d'*yylex*.

Les opérateurs '+' et '-' vont modifier la valeur dans la case pointée (*matrice[curseur]*), dans les bornes [0, 127].

Les opérateurs '>' et '<' vont modifier la valeur de *curseur* qui est le pointeur, dans les bornes [0, *MAX_CASES*].

L'opérateur ',' va proposer un prompt différent pour que l'utilisateur puisse entrer un char. L'utilisateur va pouvoir entre n'importe quel nombre de char, seul le premier sera pris en compte. Le prompt affiche un rappel de la case qui est actuellement pointée.

L'opérateur '.' affiche simplement le char de la case pointée.

Les opérateurs '/' et '/' gèrent les block.

```
1 int yyparse(void) {
2     char c;
3     int tmpfin;
4
5     while (1) {
6         switch (yylex()){
7             case '+':
8                 if (matrice[curseur] >= 127){
9                     warning("Valeur de la case hors limite ASCII");
10                    break;
11                }
12                matrice[curseur]++;
13                break;
14
15            case '-':
16                if (matrice[curseur] <= 0){
17                    warning("Valeur de la case hors limite ASCII");
18                    break;
19                }
20                matrice[curseur]--;
21                break;
22
23            case '>':
24                if (curseur >= MAX_CASES){
25                    warning("Deplacement du curseur hors limite");
26                    break;
27                }
28                curseur++;
29                break;
30
31            case '<':
32                if (curseur <= 0){
33                    warning("Deplacement du curseur hors limite");
34                    break;
35                }
36                curseur--;
37                break;
38
39            case ',':
40                printf("\t<? INPUT [%i]> ", curseur);
41
42                // lecture du premier char
43                c = getchar();
44
45                // fermer l'input sans fermer le programme
46                if (c == EOF){
47                    matrice[curseur] = 0;
48                    ungetc('\n', stdin);
49                }
50                else matrice[curseur] = c;
51
52                // Vider stdin
```

```

53         do {
54             c = getchar();
55         } while (c != '\n');
56         break;
57
58     case '.':
59         putchar(matrice[curseur]);
60         break;
61
62     case '[':
63         if((tmpfin = find_lbark()) == 0)
64             erreur("Error : missing ']' for '[' at index : %i\n",
indexligne-1);
65
66         // sauter si la case == 0
67         if (matrice[curseur] == 0){
68             indexligne = tmpfin;
69             break;
70         }
71
72         // Pas d'empilement si on est dans la meme boucle
73         if ((headstack >= 0) && (stack[headstack].debut ==
indexligne-1))
74             break;
75
76         // Pile pleine
77         if (headstack >= STACK_MAX)
78             erreur("Error : Stack overflow , limit '['...]' to : %i\n",
STACK_MAX);
79
80         // Empilement
81         headstack++;
82         stack[headstack].debut = indexligne-1;
83         stack[headstack].fin = tmpfin;
84         break;
85
86     case ']':
87         if (headstack < 0)
88             erreur("Error : missing '[' before ']' at index : %i\n",
indexligne-1);
89
90         if (matrice[curseur] != 0) {
91             indexligne = stack[headstack].debut;
92             break;
93         }
94
95         // Depilement
96         headstack--;
97         break;
98
99     case 0 :
100         return 0;
101

```

```

102         default:
103             break;
104     }
105 }
106 return 0;
107 }

```

1.1.3 Gestion des block

Afin de gérer l'imbrication des block j'ai décidé d'utiliser une structure LIFO (Last In First Out). Je crée une structure *Block* contenant 2 int, l'index du '[' et du ']' dans la ligne. Ces index vont permettre de se déplacer rapidement du début à la fin du block.

Listing 4 – Structure block

```

1 struct Block {
2     int debut;
3     int fin;
4 } typedef block;

```

Afin de matérialiser la pile j'utilise un tableau de block. Nous auront également besoin d'un pointeur *headstack* qui représente l'index du pointeur au sommet de la pile.

Listing 5 – Pile

```

1 block stack[STACK_MAX];           // pile pour garder les index des []
2 int headstack = -1;               // tete de la pile des blocks

```

J'utilise la fonction *find_lbark* pour trouver l'index du ']' fermant le block.

On part de l'index en train d'être lu et on va parcourir la ligne jusqu'à trouver le ']' fermant. Dans cas où l'on tombe sur un '[' on incrémente la variable *ignoresub*. Cette variable va permettre d'ignorer les ']' qui ne ferment pas le block actuel.

On retourne 0 si on arrive en bout de ligne sans avoir trouver le ']' fermant le block.

Listing 6 – Fonction find_lbark

```

1 int find_lbark(void){
2     int tmpindex = indexligne;
3     int ignoresub = 1;
4     char c;
5
6     while ((c = ligne[tmpindex]) != 0){
7         if (c == '[') ignoresub++;
8         if (c == ']') {
9             ignoresub--;
10            if (ignoresub == 0) return tmpindex;
11        }
12        tmpindex++;
13    }
14    return 0; // Erreur fin de ligne sans ] correspondant
15 }

```

La gestion des block se fait dans la fonction *yyparse*. Si *yylex* lit un '[', on va commencer par chercher l'index du ']' fermant.

Si *find_lbark* retourne 0, on affiche une erreur et on ferme l'interpréteur.

Selon la valeur dans la case pointée par *curseur* on saute à la fin ou on continue. Temps que l'on est dans le même block on ne rajoute pas de block supplémentaire.

Si *yylex* lit un ']' on commence par vérifier s'il y a bien une entrée dans la pile. sinon on retourne une erreur et on ferme l'interpréteur.

Selon la valeur de la case pointé à ce moment, on retourne au début du block ou on sort en dépilant au passage.

Listing 7 – Gestion des block

```
1      case '[' :
2          if ((tmpfin = find_lbark()) == 0)
3              erreur("Error : missing ']' for '[' at index : %i\n",
indexligne - 1);
4
5          // sauter si la case == 0
6          if (matrice[curseur] == 0){
7              indexligne = tmpfin;
8              break;
9          }
10
11         // Pas d'empilement si on est dans la meme boucle
12         if ((headstack >= 0) && (stack[headstack].debut ==
indexligne - 1))
13             break;
14
15         // Pile pleine
16         if (headstack >= STACK_MAX)
17             erreur("Error : Stack overflow , limit '['...]' to : %i\n",
STACK_MAX);
18
19         // Empilement
20         headstack++;
21         stack[headstack].debut = indexligne - 1;
22         stack[headstack].fin = tmpfin;
23         break;
24
25     case ']' :
26         if (headstack < 0)
27             erreur("Error : missing '[' before ']' at index : %i\n",
indexligne - 1);
28
29         if (matrice[curseur] != 0) {
30             indexligne = stack[headstack].debut;
31             break;
32         }
33
34         // Depilement
35         headstack--;
36         break;
```

1.2 Main

La fonction *main* est assez simple et s'inspire des compilateurs du cours.

Le programme n'accepte aucun argument, n'importe quel appel avec un argument va afficher un message d'aide.

Le prompt affiche la valeur du pointeur ainsi que la valeur de la case pointée.

Listing 8 – Main

```
1 int main (int argc, char* argv[]) {
2     char *s;
3
4     if (argc > 1) help(argv[0]);
5
6     printf("BrainFuck Interpreteur - version : %s\n", VERSION);
7     printf("Sortie : Ctrl + D\n");
8
9     while (1){
10         printf("\n<C:[%i] V:[%i] BFI> ", curseur, matrice[curseur]);
11
12         // Capture de la nouvelle ligne
13         s = fgets(ligne, sizeof ligne, stdin);
14         indexligne = 0;
15
16         // Fermer le programme avec un EOF
17         if (s == NULL) break;
18
19         yyparse();
20     }
21     return 0;
22 }
```

1.3 Exemple de sortie

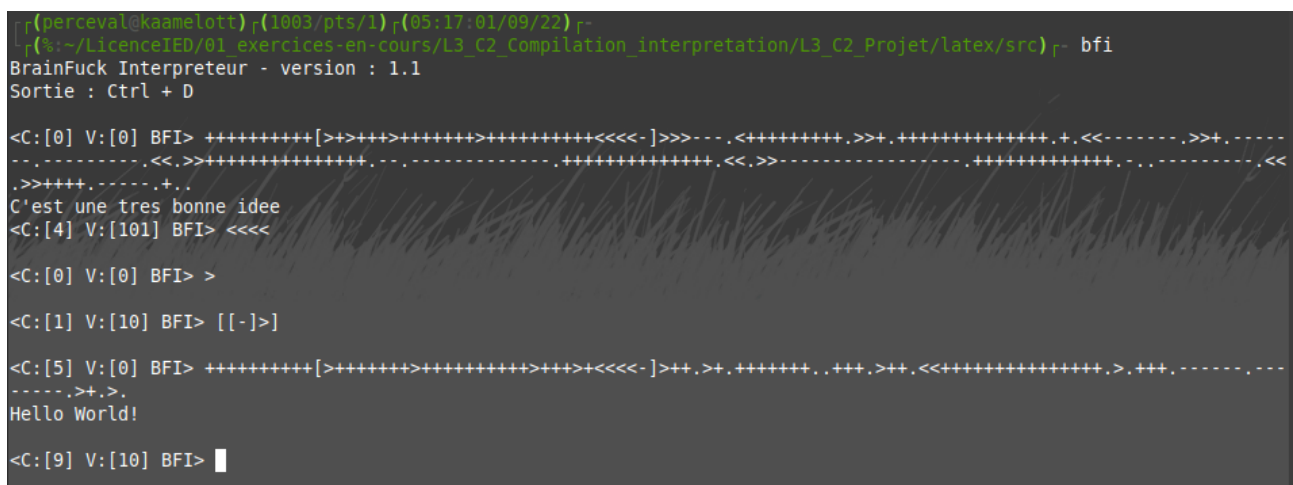


FIGURE 1 – Exemple de sortie classique


```

└─┐(perceval@kaamelott)└─┐(1004/pts/1)└─┐(05:27:01/09/22)└─┐
└─┐(%:~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/latex/src)└─┐ bfi
BrainFuck Interpreteur - version : 1.1
Sortie : Ctrl + D

<C:[0] V:[0] BFI> <
Warning: Déplacement du curseur hors limite

<C:[0] V:[0] BFI> -
Warning: Valeur de la case hors limite ASCII

<C:[0] V:[0] BFI> [+++++
Error : missing ']' for '[' at index : 0
└─┐(perceval@kaamelott)└─┐(1005 pts/1)└─┐(05:27 01/09/22)└─┐
└─┐(% ~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/latex/src)└─┐ echo $?
1

```

FIGURE 2 – Exemple de sortie avec warning et erreur

2 Logiciel libre

Cette section est pour la validation est de l'EC logiciel libre. Pour valider ce cours, il faut amener un projet au stade déployable avec un fichier `.deb` et la documentation associée.

Si vous acceptez, je souhaiterais utiliser ce projet afin de valider cet EC.

2.1 Packaging

En fouillant un peu sur Internet j'ai trouvé comment réaliser un package `.deb`.

- <http://sdz.tdct.org/sdz/creer-un-paquet-deb.html>, Tutoriel rapide pour créer un paquet debian, en expliquant la manière de créer l'arborescence.
- <http://www.linuxcertif.com/doc/Paquetage%20Debian/>, Informations sur la construction des paquets, notamment sur les conventions de nommage.

Convention de nommage Il est préférable que le paquet debian respecte la convention de nommage suivante : *nom du paquet _version-révision paquet _plateforme*.
Suivant cette convention le mon paquet se nomme : `bfi_1.1.0-2_amd64`

Arborescence Afin de réaliser le package debian, une certaine architecture est nécessaire.

```
bfi_1.1.0-2_amd64
├── DEBIAN
│   ├── control
│   └── usr
│       ├── bin
│       │   └── bfi
│       ├── share
│       │   └── man
│       │       └── man1
│       │           └── bfi.1.gz
```

Cette structure va permettre d'installer le programme `bfi` dans le répertoire `/usr/bin` et la page de manuel dans le répertoire `/usr/share/share/man/man1`.

Le répertoire `DEBIAN` contient les informations du paquet et éventuellement certains scripts de pré ou de post installation / désinstallation. Dans ce paquet on trouve uniquement les informations sur le paquets contenus dans le fichiers `control`

Fichier control Ce fichier contient le nom du paquet, la version, le type de programme, l'architecture, les dépendances, le responsable du paquet et une description.

Listing 9 – Control

```
1 Package: bfi
2 Version: 1.1
3 Section: main
4 Priority: optional
5 Architecture: amd64
6 Depends:
7 Maintainer: Martial Goehry <martial.goehry@gmail.com>
8 Description: Interpreteur pour le langage BrainFuck
```

Affichage des informations dans un prompt.

```
# dpkg -s bfi
Package: bfi
Status: install ok installed
Priority: optional
Section: main
Maintainer: Martial Goehry <martial.goehry@gmail.com>
Architecture: amd64
Version: 1.1
Description: Interpréteur pour le langage BrainFuck
```

Construction du paquet Pour construire le paquet on utilise le programme *dpkg* avec l'option *-b*

```
dpkg -b bfi_1.1.0-2_amd64
```

2.2 Man page

Pour la documentation du programme, j'ai décidé de réaliser une page de manuel.

La page de manuel *bfi* est disponible dans la section 1 (Programmes exécutables). Elle contient les sections :

- NAME : nom du programme
- SYNOPSIS : comment appeler le programme
- DESCRIPTION : Description sommaire du programme
- CODE DE RETOUR : Comportement en sortie
- ERREURS : Erreurs possibles
- NOTES : Autres informations utiles
- EXEMPLES : Exemple d'utilisation du programme
- AUTEUR : Rédacteur de la manpage.

Listing 10 – Rédaction de la page de manue

```
1 .TH bfi 1 2021-12-30 GNU "Manuel BrainFuck Interpreteur"
2
3 .SH NAME
4 .B bfi
5 – Interpreteur pour langage Brainfuck
6
7 .SH SYNOPSIS
8 .B bfi
9
10 .SH DESCRIPTION
11 Ce programme fait partie d'un projet d'Intepretation & Compilation. Cet
    interpreteur permet d'executer du code en langage BrainFuck.
12
13 Le prompt affiche l'index et la valeur de la case pointee.
14
15 L'interpreteur autorise une imbrication de block '[' ']' jusqu'a 8.
16
```

```

17 Pour sortir de l'interpreteur CTRL+D.
18
19 .SH CODE DE RETOUR
20 .TP
21 .B
22 0
23 Succes
24
25 .TP
26 .B
27 1
28 Echec
29
30
31 .SH ERREURS
32 .TS
33 tab (@);
34 | |.
35 Type@Cause
36
37 T{
38     Erreur de syntax
39 T}@Un ']' est manquant, l'erreur indique l'index du '[' solitaire
40 T{
41     Erreur de pile
42 T}@Les instructions tentent d'empiler plus de 8 block
43 T{
44     Warning pointeur
45 T}@Les instructions tentent de deplacer le pointeur hors des limites du
    tableau
46 T{
47     Warning valeur
48 T}@Les instructions tentent de modifier la valeur de la case hors des
    limites ASCII
49 .TE
50
51
52 .SH NOTES
53 .B Brainfuck
54 est un langage de programmation invente en 1993 par Urban Muller.
55
56 Avec ce programme on deplace un pointeur sur un tableau de 30.000 cases en
    modifiant eventuellement la valeur presente dans la case.
57
58 Les valeurs des cases sont comprises dans les bornes ASCII. L'affichage
    des valeurs des cases se fait au format ASCII.
59
60 Brainfuck utilise un jeu de 8 expressions:
61 .TS
62 tab (@), left, box;
63 c | c
64 cB | |.
65 Operateurs@Action

```

```

66 _
67 >@deplace le pointeur vers la case de droite
68 <@deplace le pointeur vers la case de gauche
69 +@incrémente la valeur dans la case pointée
70 -@décrémenté la valeur dans la case pointée
71 \.@affiche la valeur de la case pointée au format ASCII
72 ,@stocke dans la case pointée la valeur ASCII donner via stdin
73 [@saute au ] correspondant si l'octet pointé est égal a 0
74 ]@saute au [ correspondant si l'octet pointé est différent de 0
75 .TE

```

```

76
77
78
79 .SH EXEMPLES

```

```

80
81 .B <C:[0] V:[0] BFI>
82 ++++++++>+>+++>++++++>+++++++<<<<-]>>>+>+.++++++..+++<<+>>+++++.-----

```

```

83
84 Hello there

```

```

85
86 .B <C:[4] V:[101] BFI>

```

```

87
88
89
90 .SH AUTEUR
91 Ecrit par Martial GOEHRY <martial.goehry@gmail.com>

```

Afin de réaliser cette page de manuel, je me suis basé sur les informations de cette page :
<https://www.golinuxcloud.com/create-man-page-template-linux-with-examples/>

```

bfi(1)                                Manuel BrainFuck Interpreteur                                bfi(1)

NAME
    bfi - Interpreteur pour langage Brainfuck

SYNOPSIS
    bfi

DESCRIPTION
    Ce programme fait partie d'un projet d'Interprétation & Compilation. Cet interpreteur permet d'exécuter du
    code en langage BrainFuck.

    Le prompt affiche l'index et la valeur de la case pointée.

    L'interpréteur autorise une imbrication de block '[' ']' jusqu'à 8.

    Pour sortir de l'interpreteur CTRL+D.

CODE DE RETOUR
    0      Succès
    1      Echec

ERREURS
    Type          Cause
    -----
    Erreur de syntax  Un '[' est manquant, l'erreur indique l'index du '[' solitaire
    Erreur de pile    Les instructions tentent d'empiler plus de 8 block
    Warning pointeur  Les instructions tentent de déplacer le pointeur hors des limites du tableau
    Warning valeur    Les instructions tentent de modifier la valeur de la case hors des limites ASCII

NOTES
    Brainfuck est un langage de programmation inventé en 1993 par Urban Muller.

    Avec ce programme on déplace un pointeur sur un tableau de 30.000 cases en modifiant éventuellement la
    valeur présente dans la case.

    Les valeurs des cases sont comprises dans les bornes ASCII. L'affichage des valeurs des cases se fait au
    format ASCII.

    Brainfuck utilise un jeu de 8 expressions:

    Opérateurs      Action
    -----
    >               déplace le pointeur vers la case de droite
    <               déplace le pointeur vers la case de gauche
    +               incrémente la valeur dans la case pointée
    -               décrémente la valeur dans la case pointée
    .               affiche la valeur de la case pointée au format ASCII
    ,               stocke dans la case pointée la valeur ASCII donner via stdin
    [               saute au ] correspondant si l'octet pointé est égal à 0
    ]               saute au [ correspondant si l'octet pointé est différent de 0

EXEMPLES
    <C:[0]                                V:[0]                                BFI>
    Manual page bfi(1) line 1 (press h for help or q to quit)

```

FIGURE 3 – Affichage de man bfi

2.3 Installation désinstallation

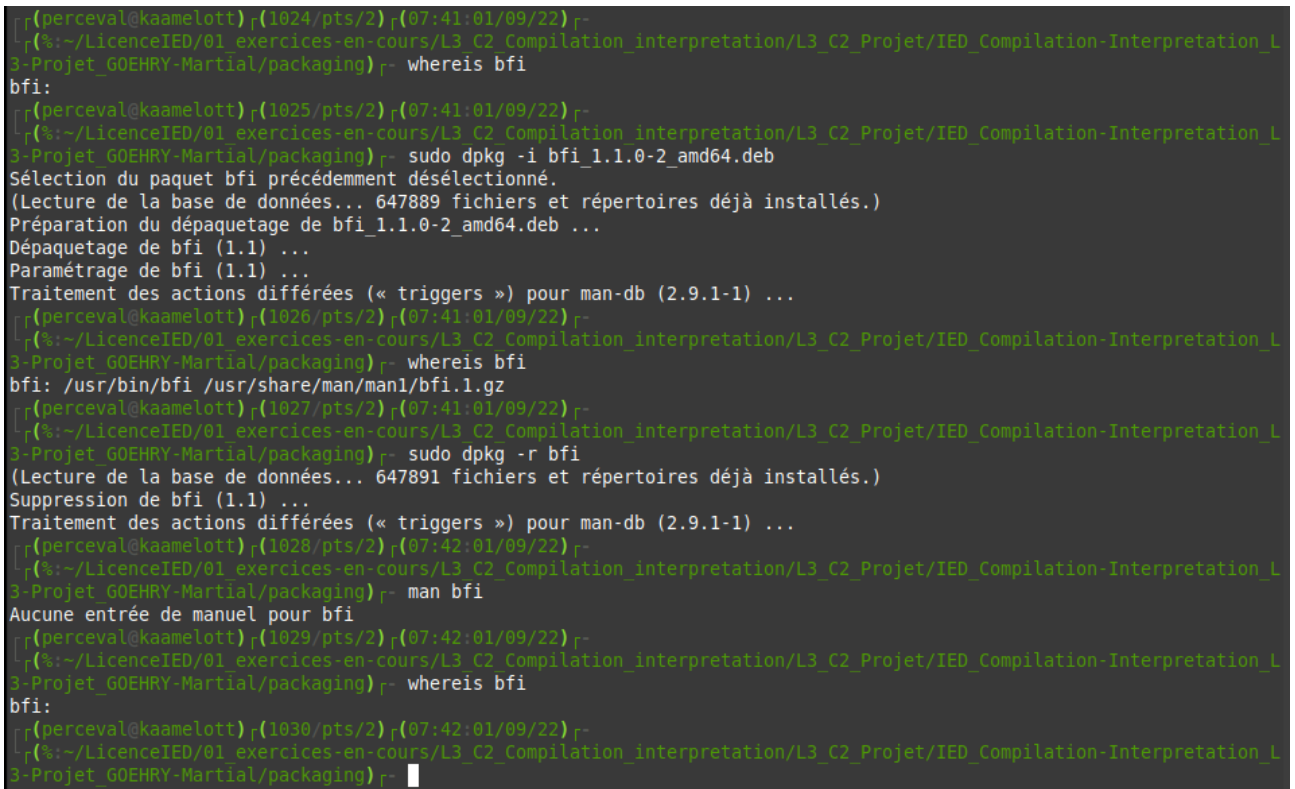
L'installation se fait avec la commande :

```
dpkg -i bfi_1.1.0-2_amd64.deb
```

La désinstallation :

```
dpkg -r bfi
```

Capture d'écran d'une installation désinstallation :



```
(perceval@kaamelott) 1024/pts/2 07:41:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - whereis bfi
bfi:
(perceval@kaamelott) 1025/pts/2 07:41:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - sudo dpkg -i bfi_1.1.0-2_amd64.deb
Sélection du paquet bfi précédemment désélectionné.
(Lecture de la base de données... 647889 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de bfi_1.1.0-2_amd64.deb ...
Dépaquetage de bfi (1.1) ...
Paramétrage de bfi (1.1) ...
Traitement des actions différées (« triggers ») pour man-db (2.9.1-1) ...
(perceval@kaamelott) 1026/pts/2 07:41:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - whereis bfi
bfi: /usr/bin/bfi /usr/share/man/man1/bfi.1.gz
(perceval@kaamelott) 1027/pts/2 07:41:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - sudo dpkg -r bfi
(Lecture de la base de données... 647891 fichiers et répertoires déjà installés.)
Suppression de bfi (1.1) ...
Traitement des actions différées (« triggers ») pour man-db (2.9.1-1) ...
(perceval@kaamelott) 1028/pts/2 07:42:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - man bfi
Aucune entrée de manuel pour bfi
(perceval@kaamelott) 1029/pts/2 07:42:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging - whereis bfi
bfi:
(perceval@kaamelott) 1030/pts/2 07:42:01/09/22 -
~/LicenceIED/01_exercices-en-cours/L3_C2_Compilation_interpretation/L3_C2_Projet/IED_Compilation-Interpretation_L
3-Projet_GOEHRM-Martial/packaging -
```

FIGURE 4 – Installation désinstallation

3 Évaluation du cours

Question 1 *Le contenu du cours a-t-il correspondu à ce que vous attendiez ? (si non, de quelle manière).*

Le cours correspondait à ce que j'attends.

Question 2 *Qu'est-ce qui vous a le plus surpris (en bien) ?*

J'ai bien aimé les chapitres sur le langage assembleur et j'ai été le plus surpris par Yacc et Bison.

Question 3 *Qu'est-ce qui vous a le plus déçu ?*

Pas de déception.

Question 4 *Quels étaient les chapitres les plus difficiles ?*

Pour moi les chapitre sur la construction de la grammaires ont été les plus dur à comprendre.

Question 5 *Quels étaient les chapitres les plus faciles ?*

Les chapitres sur le langage assembleurs ont été le plus faciles

Question 6 *Quels étaient les chapitres les plus intéressants ?*

Idem, les chapitres sur le langage assembleur. Avec un petit bonus pour le chapitre sur la génération de code.

Question 7 *Quels étaient les chapitres les moins intéressants ?*

Les chapitres sur YACC.

Question 8 *Que me suggèreriez-vous de modifier dans l'ordre des chapitres ?*

L'ordre des chapitre est cohérent. Je ne changerais rien.

Question 9 *Qu'est-ce qui est en trop dans le cours ?*

Je ne trouve pas qu'il y a des choses en trop.

Question 10 *Qu'est-ce qui manque dans le cours ?*

J'aurais aimé un peu plus d'exemple notamment pour la création de la grammaire.

Question 11 *Comment étaient les exercices du point de vue quantité (pas assez, trop).*

Les exercices sont en quantité suffisante. Le fait qu'il y ait les corrections pour les exercices non obligatoire est un vrai plus.

Question 12 *Comment étaient les exercices du point de vue difficulté (trop difficiles, trop faciles) ?*

La répartitions est correct.

Question 13 *Que donneriez-vous comme conseil à un étudiant qui va suivre le cours ?*

Je lui conseillerais de bien prendre le temps de relire les chapitres avant de se lancer dans les exercices

Question 14 *Que me donneriez-vous comme conseil en ce qui concerne le cours ?*

Je ne suis pas personnellement fan des QR code pour transmettre les liens. Afin de pouvoir ouvrir les liens, il faut : Faire une capture d'écran, utiliser un lecteur du QRcode pour enfin accéder au lien.

Question 15 *Si vous deviez mettre une note globale au cours, entre 0 et 20, laquelle mettriez-vous ?*

Je donne le note de 17/20.

Question 16 *Quelles questions manque-t-il pour évaluer correctement le cours (et bien évidemment, quelle réponse vous y apporteriez) ?*

Pas de question complémentaire.

A BrainFuck Interpréteur code complet

Listing 11 – brainfuck interpreter

```
1  /* NOM : brainfuck-interpreter.c
2   * Auteur : Goehry Martial
3   * Date : 23/11/2021
4   * Description : Interpreteur brainfuck
5   *
6   * Nombre de case de la matrice 30000
7   *
8   * Lexique :
9   * char    description
10  * >    incrementer curseur
11  * <    decrements curseur
12  * +    incrementer case pointee
13  * -    decrements case pointee
14  * .    afficher le CHAR de la case pointee
15  * ,    inserer la valeur du char donnee dans la case
16  * [    aller a l'instruction apres le ] correspondant si la case == 0
17  * ]    aller a l'instruction apres le [ correspondant si la case != 0
18  *
19  */
20
21 #include <stdio.h>
22 #include <stdlib.h>
23 #include <string.h>
24 #include <stdarg.h>
25
26 /*=====*/
27 /*          Macro          */
28 /*=====*/
29
30 #define VERSION "1.1"
31 #define MAX_CASES 30000           // Nombre de case (minimum 30 000)
32 #define MAX_LIGNE 4096           // taille maximale de la ligne
33 #define STACK_MAX 8              // taille maximale de la pile de block
34
35 /*=====*/
36 /*          Structure Stack          */
37 /*=====*/
38 // conservation en memoire des '[' ]
39 struct Block {
40     int debut;
41     int fin;
42 } typedef block;
43
44 /*=====*/
45 /*          Variables Globales          */
46 /*=====*/
47
48 int yylval = 0;                  // Valeur d'une operation
49
50 char ligne[MAX_LIGNE] = {0};    // ligne lue
```

```

51  int indexligne;                // index du prochain char a traiter
52
53  unsigned char matrice[MAX_CASES] = {0}; // tableau des cases pour
    brainfuck
54  int curseur = 0;                // index de la case utilisee
55
56  char *ops = "><+-.,]";          // Operateurs acceptes
57
58  block stack[STACK_MAX];          // pile pour garder les index des []
59  int headstack = -1;              // tete de la pile des blocks
60
61
62  /*=====*/
63  /*          Fonctions          */
64  /*=====*/
65
66  // erreur – sortie en cas d'erreur
67  void erreur(char* message, ...){
68      va_list args;
69      va_start(args, message);
70      vfprintf(stderr, message, args);
71      va_end(args);
72      exit(1);
73  }
74
75  // warning – avertissement de l'interpreteur
76  void warning(char* message){
77      fprintf(stderr, "Warning: %s\n", message);
78  }
79
80
81  // find_lbark – trouver la ] correspondante
82  int find_lbark(void){
83      int tmpindex = indexligne;
84      int ignoresub = 1;
85      char c;
86
87      while ((c = ligne[tmpindex]) != 0){
88          if (c == '[') ignoresub++;
89          if (c == ']') {
90              ignoresub--;
91              if (ignoresub == 0) return tmpindex;
92          }
93          tmpindex++;
94      }
95      return 0;    // Erreur fin de ligne sans ] correspondant
96  }
97
98
99  // yylex – Lecture des operations
100 int yylex(void) {
101
102      // Recommencer a la fin de la ligne

```

```

103     if (ligne[indexligne] == '\n'){
104         return 0;
105     }
106
107     // Retirer tous les char qui ne sont pas des operateurs
108     while (!(strchr(ops, ligne[indexligne])))
109         indexligne++;
110
111     // Recuperation de l'operateur
112     yylval = ligne[indexligne++];
113     return yylval;
114 }
115
116 // yyparse — analyseur lexical
117 int yyparse(void) {
118     char c;
119     int tmpfin;
120
121     while (1) {
122         switch (yylex()){
123             case '+':
124                 if (matrice[curseur] >= 127){
125                     warning("Valeur de la case hors limite ASCII");
126                     break;
127                 }
128                 matrice[curseur]++;
129                 break;
130
131             case '-':
132                 if (matrice[curseur] <= 0){
133                     warning("Valeur de la case hors limite ASCII");
134                     break;
135                 }
136                 matrice[curseur]--;
137                 break;
138
139             case '>':
140                 if (curseur >= MAX_CASES){
141                     warning("Deplacement du curseur hors limite");
142                     break;
143                 }
144                 curseur++;
145                 break;
146
147             case '<':
148                 if (curseur <= 0){
149                     warning("Deplacement du curseur hors limite");
150                     break;
151                 }
152                 curseur--;
153                 break;
154
155

```

```

156     case ',':
157         printf("\t<? INPUT [%i]> ", curseur);
158
159         // lecture du premier char
160         c = getchar();
161
162         // fermer l'input sans fermer le programme
163         if (c == EOF){
164             matrice[curseur] = 0;
165             ungetc('\n', stdin);
166         }
167         else matrice[curseur] = c;
168
169         // Vider stdin
170         do {
171             c = getchar();
172         } while (c != '\n');
173         break;
174
175     case '.':
176         putchar(matrice[curseur]);
177         break;
178
179     case '[':
180         if ((tmpfin = find_lbark()) == 0)
181             erreur("Error : missing ']' for '[' at index : %i\n",
indexligne - 1);
182
183         // sauter si la case == 0
184         if (matrice[curseur] == 0){
185             indexligne = tmpfin;
186             break;
187         }
188
189         // Pas d'empilement si on est dans la meme boucle
190         if ((headstack >= 0) && (stack[headstack].debut ==
indexligne - 1))
191             break;
192
193         // Pile pleine
194         if (headstack >= STACK_MAX)
195             erreur("Error : Stack overflow , limit '['[...] ' to : %i\
n", STACK_MAX);
196
197         // Empilement
198         headstack++;
199         stack[headstack].debut = indexligne - 1;
200         stack[headstack].fin = tmpfin;
201         break;
202
203     case ']':
204         if (headstack < 0)

```

```

205         erreur("Error : missing '[' before ']' at index : %i\n", indexligne-1);
206
207         if (matrice[curseur] != 0) {
208             indexligne = stack[headstack].debut;
209             break;
210         }
211
212         // Depilement
213         headstack--;
214         break;
215
216     case 0 :
217         return 0;
218
219     default:
220         break;
221     }
222 }
223 return 0;
224 }
225
226 void help (char *program){
227     printf("Usage: %s\n\n", program);
228     printf("Interpreteur en ligne de commande pour le langage BrainFuck\n\n");
229     printf("
+++++>++++>+++++>+++++<<<<->>>>+++++..++++<<++++>>++++.
");
230     exit(0);
231 }
232
233 /*=====*/
234 /*      MAIN      */
235 /*=====*/
236
237 int main (int argc, char* argv[]) {
238     char *s;
239
240     if (argc > 1) help(argv[0]);
241
242     printf("BrainFuck Interpreteur - version : %s\n", VERSION);
243     printf("Sortie : Ctrl + D\n");
244
245     while (1){
246         printf("\n<C:[%i] V:[%i] BFI> ", curseur, matrice[curseur]);
247
248         // Capture de la nouvelle ligne
249         s = fgets(ligne, sizeof ligne, stdin);
250         indexligne = 0;
251
252         // Fermer le programme avec un EOF
253         if (s == NULL) break;

```

```
254
255     yyparse();
256 }
257 return 0;
258 }
```
